A Robust 2-Refinement Algorithm in Octree and Rhombic Dodecahedral Tree Based All-Hexahedral Mesh Generation

Yongjie Zhang[1,⋆], Xinghua Liang[1], and Guoliang Xu[2]

[1] Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA
[2] LSEC, Institute of Computational Mathematics, Academy of Mathematics and System Sciences, Chinese Academy of Sciences, Beijing 100190, China

Summary. In this paper, we present a novel 2-refinement algorithm for adaptive all-hexahedral mesh generation based on two tree structures: octree and rhombic dodecahedral tree. Given a smooth boundary surface, we first use a pre-defined error function to detect the main surface features, and build a strongly-balanced octree. Then a novel 2-refinement algorithm is developed to eliminate all hanging nodes in the octree, which is robust for any unstructured meshes and induces a smooth transition with very little propagation. Later, all elements outside and around the boundary are removed to create the octree core mesh and a buffer zone. The boundary points on the core mesh are projected onto the surface and form the final mesh. Motivated from nature, a new tree structure based on rhombic dodecahedron is introduced. Sharp features are also detected and preserved during mesh generation. Finally, pillowing, geometric flow and optimization-based smoothing are applied to improve quality of the constructed meshes.

Key words: 2-refinement, all-hexahedral mesh, octree, rhombic dodecahedron, sharp feature.

1 Introduction

In finite element analysis, unstructured hexahedral (hex) meshes are by far preferred due to their superior performance over tetrahedral meshes in terms of smaller element counts, increased accuracy and improved reliability. However, there are only a few algorithms developed in the literature for all-hex mesh generation. Among them, sweeping [7], paving/plastering [13] and

⋆ Corresponding author. Tel: (412) 268-5332; Fax: (412) 268-3348; Email: *jessicaz@andrew.cmu.edu* (Y. Zhang).

whisker weaving [4] are not fully automatic and they require user interactions. As a promising solution, the grid-based method is widely used due to its robustness and effectiveness.

In the grid-based method, a fitted 3D grid of hexes using octree is constructed, and then additional hexes are added at the boundaries to fill gaps [11]. 2- and 3-refinement templates were developed for adaptive mesh generation. 2-refinement uniformly subdivides a selected hex into 8 smaller ones, while 3-refinement results in a 1-to-27 split. Due to the ease of implementation, 3-refinement templates [12, 14] were studied first. However, 3-refinement produces much more new elements and the transition region is not so smooth as 2-refinement [11, 12]. In addition, 2-refinement yields better aspect ratio than 3-refinement. The 2-refinement method has been thoroughly studied in 2D quadrilateral mesh generation [8, 9]. However, the implementation of 2-refinement in 3D is still a challenge. The 2-refinement method was first introduced for structured hex meshes only [3, 6, 12], and was improved later using pillowing [2]. All these developments have difficulty in handling situations where two or more refined regions are adjacent to each other. They require a great amount of propagation and cannot deal with concavity.

In this paper, we introduce a novel 2-refinement algorithm for unstructured all-hex meshes. Given a smooth boundary surface, four steps are designed to construct adaptive hex meshes based on two tree structures: octree and rhombic dodecahedral (RD) tree. The key contributions of our work include: (1) a novel 2-refinement method which is robust for any unstructured meshes and yields a smooth transition with very little propagation. 3-refinement and 2-refinement are compared in detail; and (2) a novel rhombic dodecahedral (RD) tree structure is introduced for all-hex mesh construction. Moreover, sharp features are preserved and the mesh quality is improved using pillowing, geometric flow and optimization-based smoothing. We have applied our algorithm to several complicated geometries. Our algorithm is able to efficiently capture the main details and sharp features (if have), and generates meshes with good quality.

The remainder of this paper is organized as follows: Section 2 explains the detailed algorithm for octree-based hex mesh generation using 2-refinement. Section 3 introduces a new RD tree structure. Section 4 discusses sharp feature preservation and mesh quality improvement. Section 5 shows some application results. Finally, Section 6 presents our conclusion.

2 Octree-Based Hex Mesh Generation

Given a closed smooth surface mesh as input, we design four steps to generate adaptive all-hex meshes based on octree: adaptive octree construction, hanging node elimination via 2-refinement, buffer zone clearance, and projection.

2.1 Adaptive Octree Construction

As the first step, a cube is constructed which bounds the given surface mesh. This cube is the *root* of the octree, as marked as level 0. Cells obtained after refining the i^{th}-level cell will be marked as level $(i+1)$. To detect surface features, we introduce a feature sensitive error function [15], $ERROR = \sum_{i=1}^{27} \frac{|f^{i+1}(P)-f^{i}(P)|}{|\nabla f^{i}(P)|}$, where $f^{i}(P)$ is the distance from node P at level i to the surface. A total of 27 nodes need to be measured for each cell. For level i, the function values of 12 edge middle points, 6 face middle points and 1 center point can be obtained through a trilinear interpolation. This error function estimates the difference of the isosurface between two neighboring levels. Given an error tolerance ε, we refine cells with a larger error ($> \varepsilon$). In addition, to generate meshes with good aspect ratio, we limit the level difference between two adjacent cells to be less than or equal to one. In the end, a strongly balanced octree is obtained, see Fig. 1(a).

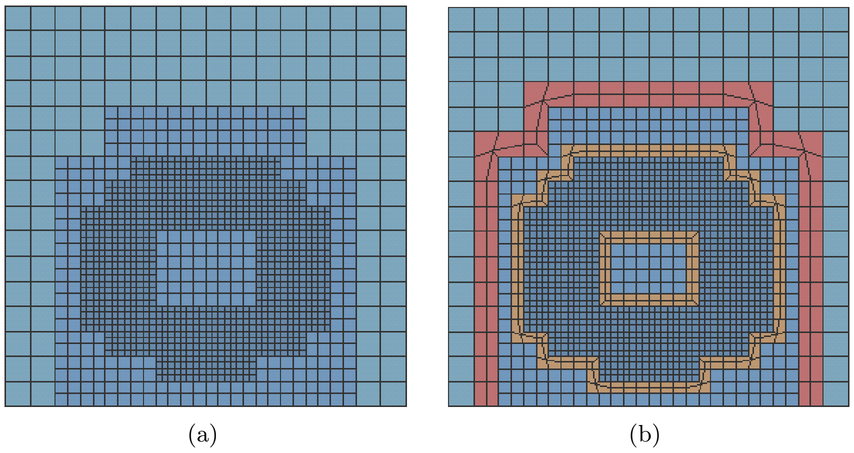

Fig. 1. (a) A strongly balanced octree; and (b) the obtained result after pillowing.

2.2 Robust 2-Refinement for Hanging Node Elimination

Among the existing solutions, template-based methods such as 3- and 2-refinement are the most widely used to remove hanging nodes inside the adaptive octree. In the 3-refinement algorithm [12, 14], each node is checked and marked whether it needs to be refined according to a pre-defined error function. Since there are eight nodes in one hexahedron, there are $2^8 = 256$ possible configurations. Considering symmetry and complementary, five distinct templates were summarized [14]. Each hexahedron belonging to them are applied with one of the templates. The remaining hexahedra are then converted to one of them according to a look-up table. This conversion is repeated until no more propagation is needed. Finally an adaptive octree without hanging nodes is constructed. As we can see, 3-refinement is easy to implement. However, when compared to 2-refinement which only produces 8 smaller hexahedra, a hexahedron using 3-refinement will be converted to 27 smaller ones. Obviously, 3-refinement produces much more new elements and the transition region is not so smooth as 2-refinement. In addition, 2-refinement yields better aspect ratio than 3-refinement.

The implementation of 2-refinement in 3D is still a challenge. The main difficulty is how to limit the propagation during hanging node elimination. Very few work has been done and they were limited to structured meshes only [3, 11]. One attempt for an unstructured mesh was developed in [2], but it requires an overall refinement of all elements beforehand, which increases the element number rapidly by 7 times. Recently, Qian and Zhang [10] attempted to apply 2-refinement to unstructured meshes. In this approach, given a uniform unstructured mesh, some core regions are defined, then three steps are adopted: refine the core region until the requirement for 2-refinement is satisfied, split the transition layer into two layers such that each transition element has only one transition face, and finally remove hanging nodes. This method can generate adaptive hex meshes from uniform unstructured ones, and provide smooth transition layers. However, the first step of this approach involves a great amount of propagation, which sometimes may generate more elements than using 3-refinement. Moreover, the second step can only work for situations where all the refined regions are isolated.

In this paper we introduce a robust 2-refinement algorithm to remove hanging nodes in the adaptive octree, which needs very little propagation. Here are several definitions used in the following algorithm description.

Transition element: *A transition element is an element connecting elements at two different levels.*
Transition face: *A transition face is a face in a transition element which is also shared by an element at a lower level.*
Transition node: *A node on a transition face is named a transition node.*
Non-manifold transition region: *A non-manifold transition region is a region where two or more refined regions are adjacent to each other. Otherwise, if all the refined regions are isolated, it is called a "manifold transition region".*

Before applying the 2-refinement templates introduced by Schneiders [11, 12], as shown in Fig. 2, to unstructured all-hex meshes, we need to solve the following two problems: (1) The coupling of transition elements. The 2-refinement template in Fig. 2(a) is applied to a block of four transition elements sharing an edge, see Fig. 2(b). We need to ensure that there is such a block for each transition element, especially for unstructured meshes with arbitrary valence number. Then an efficient way to implement the 2-refinement template must be developed since it has to be flipped for each pair of ele-

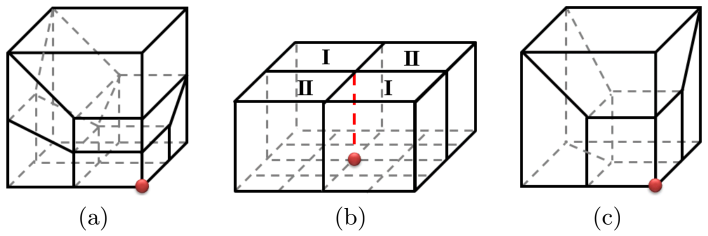

Fig. 2. 2-refinement templates for octree [11, 12]. (a) 2-refinement templates; (b) the way to apply 2-refinement templates; and (c) another template to remove the remaining hanging nodes.

ments sharing a face in that block (marked as "I" and "II" in Fig. 2(b)). (2) The concavity in the octree in which there are elements with more than one transition face. There is no template to handle such transition elements. This problem exists in 3-refinement methods as well.

The first problem can be easily solved by refining all the elements surrounding each irregular transition node in the adaptive octree. However, it is not so easy to solve the second one. This can be achieved by isolating the refined region completely using the pillowing technique, which duplicates the corresponding transition nodes for the transition elements and inserts new layers. After that, each transition element will have one and only one transition face. The pillowing procedure duplicates each transition node along the average normal direction of its neighboring transition faces. The distance between a node and its duplicate is half the minimum length of its neighboring edges, as shown in Fig. 3. After the isolation of the refined regions, the template in Fig. 2(a) is utilized to eliminate hanging nodes.

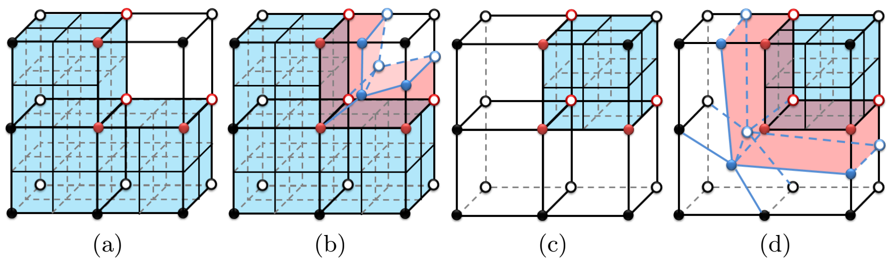

Fig. 3. Pillowing for concave (a-b) and convex (c-d) cases. Solid nodes are nodes on the boundary of the refined region, circles are interior to the refined region. Red points are transition nodes, and blue points are duplicated nodes. Blue region is the refined region, and pink region is the pillowed layer.

During pillowing, we need to pay extreme attention to non-manifold transition regions. For an octree, these non-manifold cases happen in a 8-element

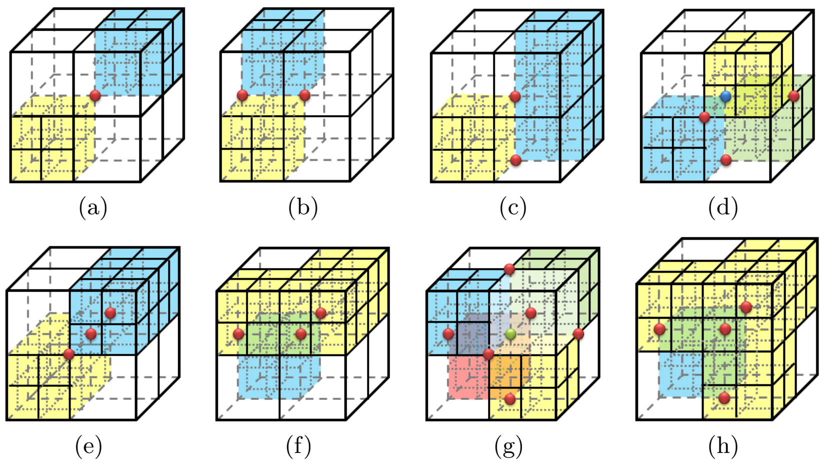

Fig. 4. Pillowing for non-manifold transition regions. Colored elements are at level i, and others are at level $(i+1)$. Red, blue and green points are duplicated two, three and four times, respectively.

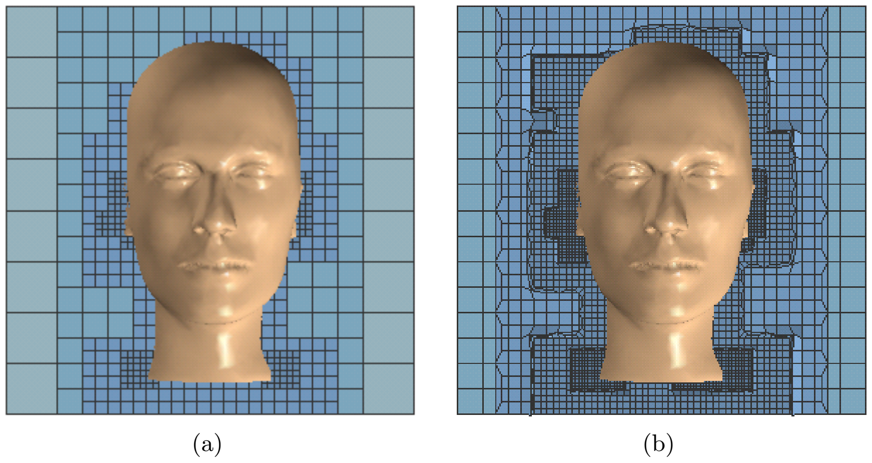

Fig. 5. (a) Adaptive octree with hanging nodes; and (b) hanging nodes are removed via 2-refinement.

block, in which each element may be refined (at level $i+1$) or not refined (at level i). Obviously, there are $2^8 = 256$ possible configurations. Out of these configurations, only the blocks with $2 \sim 6$ refined elements may have non-manifold cases. Considering symmetry and complementary, eight distinct cases are summarized, which can be overcome by duplicating the non-manifold nodes, as shown in Fig. 4. Colored elements are at level i, and others are at level $(i+1)$. During pillowing, we duplicate the non-manifold node m times,

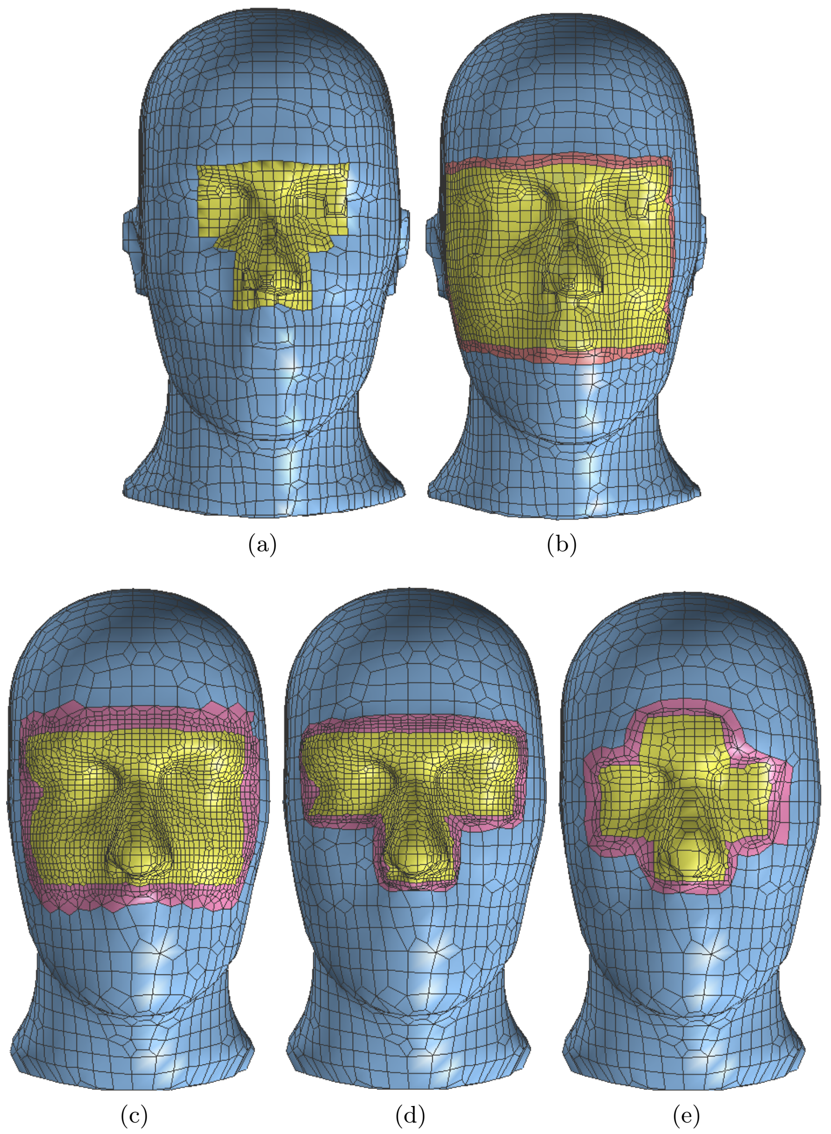

Fig. 6. Comparison between 2- and 3-refinement. (a, b) 2-refinement method introduced in [10], before (a) and after (b) removing hanging nodes. Mesh sizes are (9,503; 7,545) and (16,052; 13,868), respectively; (c) 3-refinement, with a mesh size of (14,278; 12,442); (d) 3-refinement with pillowing, with a mesh size of (12,141; 10,464); and (e) our 2-refinement method, with a mesh size of (10,342; 9,098). In "$(m;n)$", m is the vertex number and n is the element number. Yellow is the refined region, and pink is the transition region. Note that the yellow region in (a) is the initial refined region for (b-e).

where m is the number of refined regions surrounding it. For example, the red, blue and green points need to be duplicated two, three and four times, respectively. For the remaining cases, the elements at level i need to be refined such that they are converted to the above solvable ones. Fig. 1(b) shows a pillowing result in 2D, and Fig. 5 shows the adaptive octree of a head model before and after removing hanging nodes.

As a comparison, we choose a head mesh in Fig. 6 as an example, only refining the eyes and the nose using 2- and 3-refinement. The 2-refinement method proposed in [10] requires an initial uniform hexahedral mesh with the specified refined regions, as show in Fig. 6(a). The result after eliminating all the hanging nodes is shown in Fig. 6(b). We can see that a great amount of propagation is needed for this method (see the large yellow region), and the resulting mesh is not symmetric. In 3-refinement, the templates can only handle elements with one transition edge or face. If there are elements with two or more transition faces, which normally form concavities, then a refinement is needed and it is very easy to propagate to a large region, see the yellow region in Fig. 6(c). This drawback can be overcome by pillowing, see Fig. 6(d), but it will decrease the mesh quality. Fig. 6(e) is the result of our 2-refinement method, which is the best of the five results. It can provide a smooth transition between different levels with very little propagation. In addition, our method introduces the fewest number of new nodes and elements.

2.3 Buffer Zone Clearance

After generating the adaptive octree, we delete elements outside or close to the boundary surface to obtain a hexahedral core mesh. We call such a procedure *buffer zone clearance*. For example, if the shortest distance from any vertex to the boundary is less than a pre-defined threshold ε_s, all elements sharing this vertex are deleted. Here we choose $\varepsilon_s = \frac{1}{2}max(s_i)$, where s_i is the size of the i^{th} element sharing this vertex. To generate good-quality elements around the boundary, we design the following two operations to improve the boundary of the core mesh:

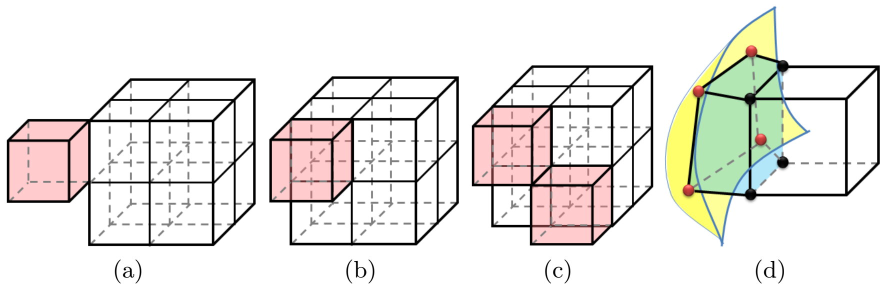

Fig. 7. Buffer zone clearance and projection to the surface. (a-c) Three cases in buffer zone clearance; and (d) the buffer layer construction.

(1) Delete single elements which only share a point, an edge or a face with other elements, as shown in Fig. 7(a&b). If not doing so, these elements may induce incorrect connectivity during the following projection procedure and hinder mesh quality improvement.

(2) Delete elements that have non-manifold connectivity on the boundary, as shown in Fig. 7(c). Again, these elements may induce wrong connectivity when we project boundary nodes to the surface.

2.4 Projection

As the last step, we project all the boundary points of the core mesh to the surface. Then the buffer layer is generated by connecting the boundary points and their corresponding projection points, see Fig. 7(d). To obtain elements with better quality, the buffer layer can be split into two layers such that more freedoms are provided for quality optimization later. Fig. 8(a) shows an example of buffer zone clearance, Fig. 8(b) is the result after projection. Obviously, the mesh quality needs to be improved (see Section 4).

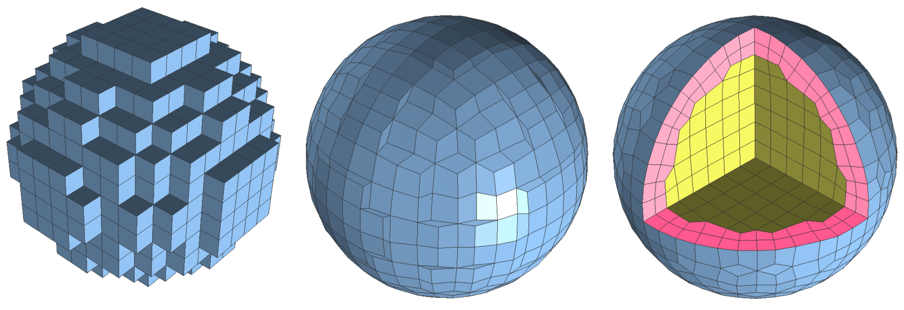

Fig. 8. Buffer zone clearance and projection. (a) The core mesh after buffer zone clearance; (b) the mesh after projection; and (c) the final mesh after smoothing. Yellow is the core mesh, and pink is the two buffer layers.

3 RD-Tree Based Mesh Generation

Besides cubes, the rhombic dodecahedron (RD) in Fig. 9(a) can be used to tessellate 3D space. The RD structure naturally exists in the world. Honeycomb consists of tessellating cells, each of which is a hexagonal prism capped with a half RD; some minerals like garnet form a RD crystal habit; and the RD structure appears in the unit cells of diamond as well. As shown in Fig. 9(a), a RD has 14 nodes, 24 equilong edges and 12 rhombic faces. For all the faces, the dihedral angle is 120°. Moreover, by adding a center point, a RD can be split into four identical rhombic hexahedra by two ways, as shown in Fig. 9(b & c).

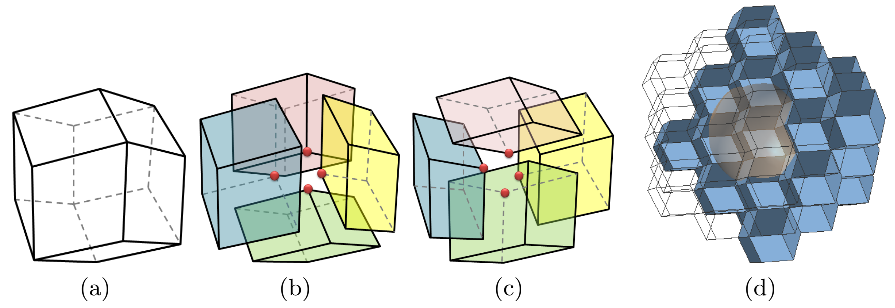

Fig. 9. Rhombic dodecahedron (a) and its two decompositions (b-c). (d) A uniform RD tree.

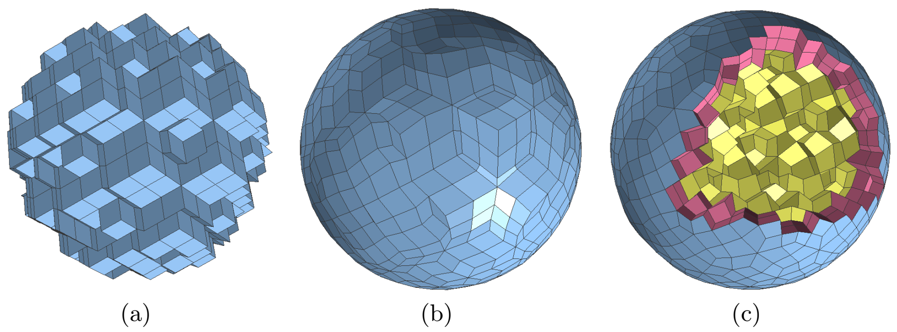

Fig. 10. The meshing results of a sphere using the RD tree. (a) The result after buffer zone clearance; (b) the result after projection; and (c) the final mesh after smoothing. Yellow is the core mesh, and pink is the two buffer layers.

Different from using a bounding box for the octree-based method, here a uniform RD tree is built by tessellating RD elements to cover the whole surface mesh, see Fig. 9(d). The RD tree is then converted to a hex tree by splitting all the RD elements into hexes. During splitting, for each RD element we check the valence of each vertex and choose the template in Fig. 9(b-c) which minimizes the overall valence number for the final hex tree, see Table 1. It is obvious that the hex tree contains a lot of irregular points with a valence number other than eight. Fig. 10 shows the meshing results of a sphere based on the RD tree. By comparing with Fig. 8, we can observe that using RD tree can provide more unstructured elements with a lot of irregular nodes, and the elements follow different orientations.

For an adaptive tree based on RD, each rhombic hex is refined based on the feature sensitive function, and then pillowing and 2-refinement are applied to eliminate hanging nodes in the RD-based hex tree (see Fig. 11). Note that the RD tree is unstructured, the pillowing method in Section 2.2 cannot be directly applied. Here a generalized method is developed. All elements at the lower

Table 1: Statistics of valence number before and after splitting optimization.

Number of Valence	1	2	3	4	5	6	7	8	9	10	12
One-direction splitting	112	546	172	1647	0	420	0	0	60	216	1334
Optimized splitting	110	441	133	1284	35	599	47	565	19	534	740

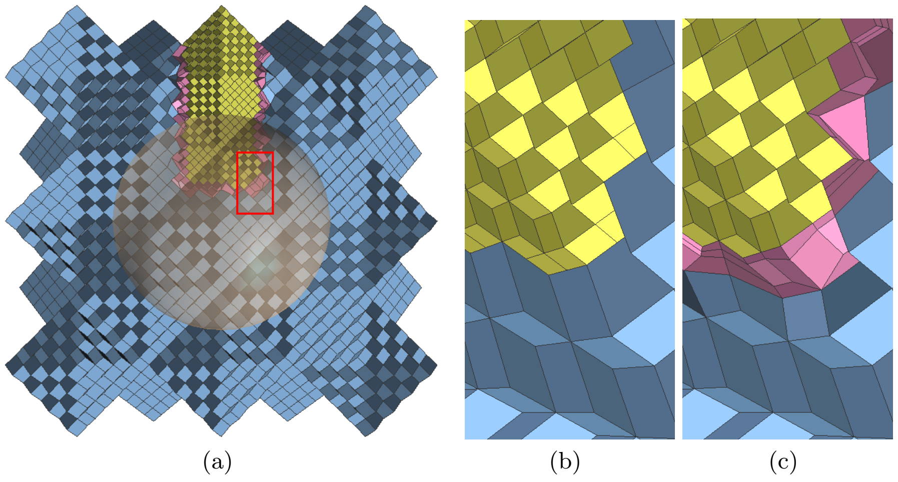

Fig. 11. RD-based adaptive hex tree construction. (a) Adaptive hex tree after pillowing and 2-refinement; and (b-c) zoom-in pictures of (a) before (b) and after (c) pillowing and 2-refinement.

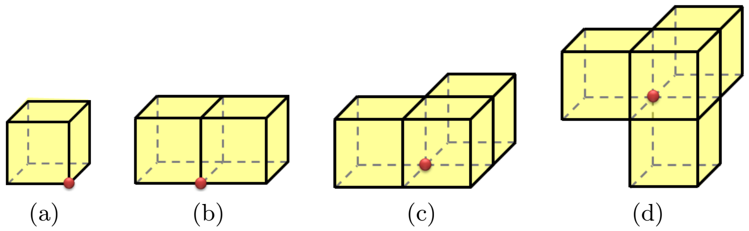

Fig. 12. Four pillowable patches. The red node is a transition node.

level surrounding a transition node are classified into different patches based on their connectivity, such that each patch is manifold. Four kinds of patches, as shown in Fig. 12, can be pillowed easily. The remaining situations need to be refined to make them pillowable. This procedure induces propagation in the RD tree. When pillowing is done, 2-refinement templates are applied to eliminate all the hanging nodes, see Fig. 11(c). The following buffer zone clearance and projection procedures are similar to the octree-based method. Fig. 13 shows two adaptive all-hex meshes of the head model based on the octree and the RD tree, respectively.

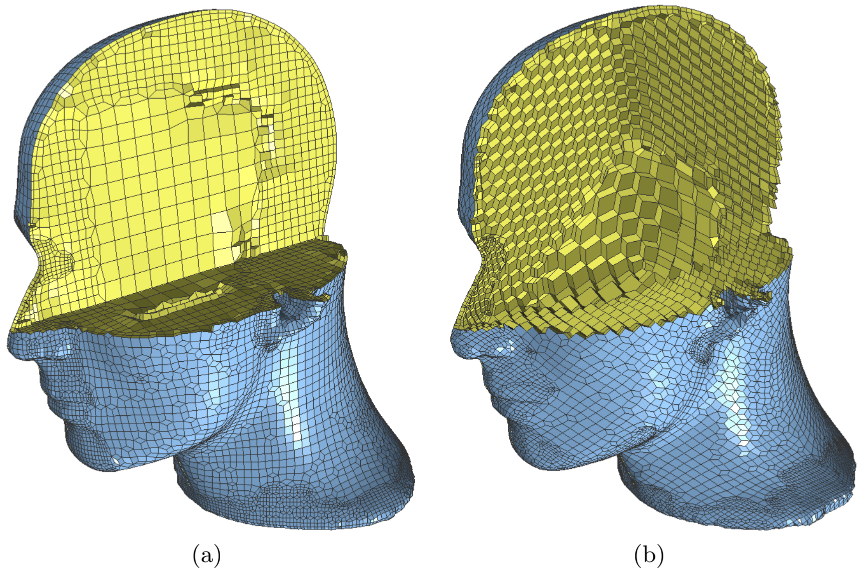

Fig. 13. Adaptive all-hex meshes of the head model. (a) The octree-based method; and (b) the RD tree based method.

Remark: For both octree and the RD tree, our 2-refinement algorithm introduces very little propagation in eliminating hanging nodes. This is because the pillowing method can efficiently isolate transition faces, and thus it is able to handle non-manifold transition regions. From these two tree structures, we can observe that our 2-refinement algorithm is robust for not only structured meshes but also unstructured ones with a lot of irregular nodes.

4 Sharp Feature Preservation and Quality Improvement

A lot of input surface meshes, such as CAD models, contain sharp features which are important and cannot be neglected. The sharp feature preservation algorithm has been proposed in [10]. We suppose all the sharp features are provided along with the given surface mesh, see Fig. 14(a). We firstly identify each joint point P_{joint} shared by multiple sharp curves, and find the closest node in the generated mesh. Each sharp curve has two joint points, P_{start} and P_{end}. A shortest path is found between them using Dijkstra's algorithm [1], and then each node on this path is projected to the sharp curve. Three criteria are used to set the priority: (1) For sharp curves, the longer one is preserved first; (2) for joint points, the node whose adjacent edge forms a smaller angle to the sharp curve is preferred; and (3) for nodes on the path, the node with a minimal projection distance is chosen. Fig. 14 shows a result for sharp feature preservation.

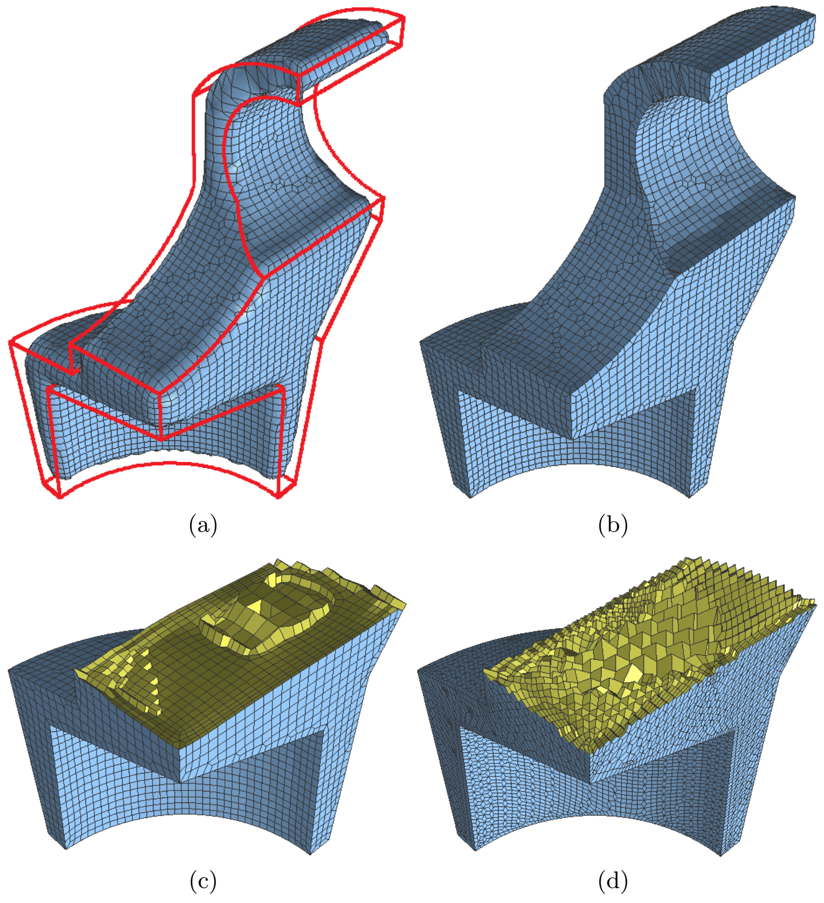

Fig. 14. Sharp feature preservation for the hook model. (a) The input sharp curves, and the generated smooth mesh using the octree-based method in Section 2; (b) the final mesh with sharp feature preservation; (c) cross section of the final mesh based on octree; and (d) cross section of the final mesh based on the RD tree.

Quality improvement is important for mesh generation. Here we choose the scaled Jacobian to measure the mesh quality [16]. For each node x in a hex, three edge vectors are defined as $e_i = x_i - x$ $(i = 1, 2, 3)$. Then the Jacobian matrix is defined as $J = [e_1, e_2, e_3]$, and *Jacobian* is defined as $Jacobian(x) = det(J)$. If e_1, e_2 and e_3 are normalized, $det(J)$ is also called the *scaled Jacobian*. For mesh quality improvement, geometric flow is first applied to improve the overall quality of the mesh [16, 17]. Then, optimization-based smoothing is adopted to improve the worst quality element of the mesh [5]. The combination of geometric flow and optimization-based smoothing can generally result in good-quality for smooth hexahedral meshes. However, for meshes with sharp features, pillowing [10] is required which can guarantee that there are no element with more than two edges lying on the sharp curves, and no element with more than one face lying on the same surface patch.

5 Results and Discussion

Several models are used to test our algorithm, including two smooth head models, see Figs. 13 & 15, the Buddha, see Fig. 16, and three CAD models with sharp features, see Figs. 14, 17 and 18. We use both the octree and the RD tree to generate adaptive all-hex meshes so that we can compare these two different tree structures. Our results were computed on a PC equipped with a 2.93 GHz Intel X3470 CPU and 8GB of Memory.

Statistics of these meshes are given in Table 2. After applying quality improvement techniques, all the hexahedral meshes are in reasonable good quality. The pillowing plus 2-refinement algorithm is applied to all the models. Although there are various kinds of non-manifold transition regions in these meshes, our approach is able to handle all of them with only a small amount of propagation. Moreover, the results also indicate that our feature sensitive function can effectively capture important features on the surface, such as the nose and eyes in these two head models.

Our algorithm also works for CAD models, as shown in Figs. 14, 17 and 18. For CAD models, we restrict the surface meshes to be uniform and only the interior region of these models are adaptive. Because pillowing is adopted in our algorithm, every surface element in these meshes has at most two edges lying on the same curve, and at most one surface lying on the same surface patch, which leads to good quality meshes.

By comparing the octree-based and RD tree based methods, we can observe that octree provides more structured elements, and the transition between two different levels are smoother. Therefore, the octree-based meshes are preferred in finite element analysis. However, the RD tree may be better for some specific applications. For example, the RD tree structure can better represent the honeycomb and some minerals like garnet.

6 Conclusions

In this paper, we present a novel algorithm which can generate unstructured adaptive all-hex meshes using two tree structures and 2-refinement. For any given smooth surface, four steps are designed to construct adaptive hex meshes with reasonable good quality. Compared to 3-refinement and other 2-refinement approaches, our 2-refinement method introduces very little propagation and is capable of handling complicated non-manifold transition regions. Moreover, a new RD tree structure is introduced besides the octree.

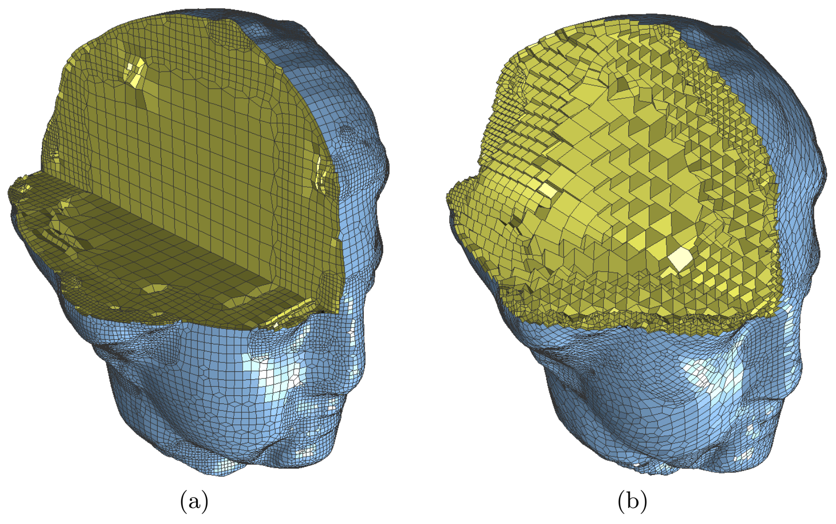

Fig. 15. Adaptive all-hex meshes of the Igea model. (a) The octree-based method; and (b) the RD tree based method.

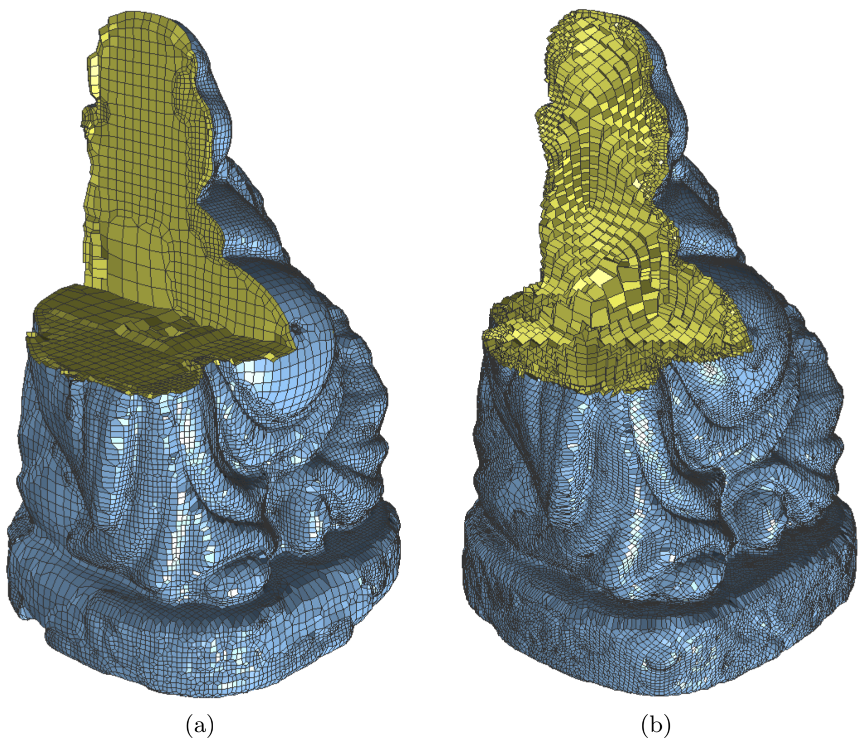

Fig. 16. Adaptive all-hex meshes of the Buddha model. (a) The octree-based method; and (b) the RD tree based method.

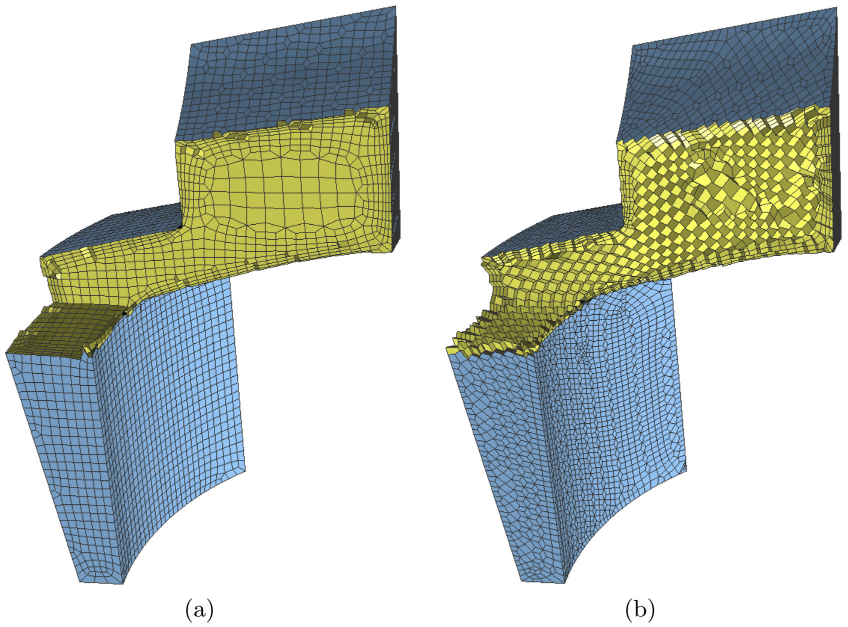

Fig. 17. Adaptive all-hex meshes of the Hook2 model. (a) The octree-based method; and (b) the RD tree based method.

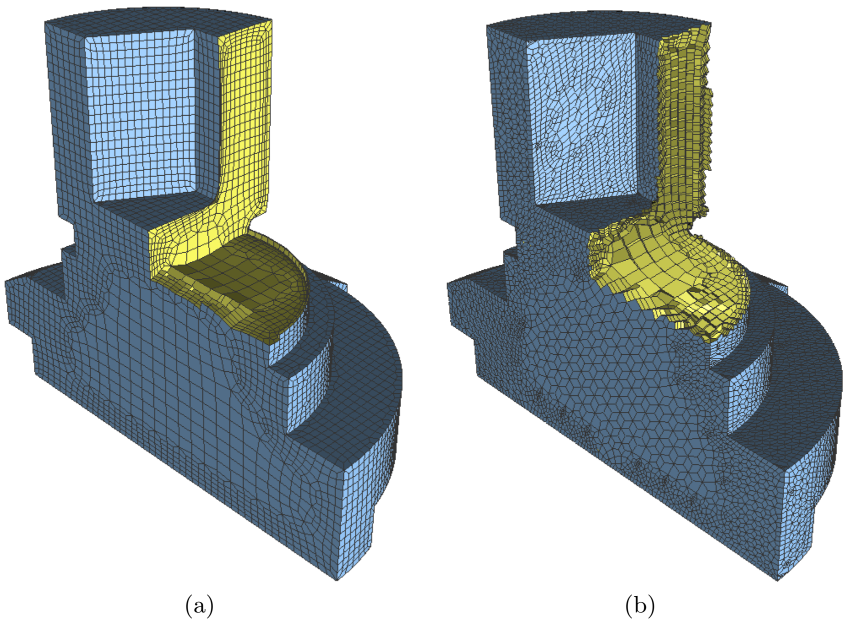

Fig. 18. Adaptive all-hex meshes of the Varco3 model. (a) The octree-based method; and (b) the RD tree based method.

Table 2: Mesh statistics of the resulting models.

Model	Method	Mesh Size (vertex; element)	Scaled Jacobian [worst; best]	Time (s)
Head	Octree	(64,258; 56,419)	[0.013; 1.0]	169
	RD tree	(51,985; 45,336)	[0.016; 1.0]	95
Igea	Octree	(110,352; 98,087)	[0.023; 1.0]	237
	RD tree	(74,012; 65,198)	[0.017; 1.0]	122
Buddha	Octree	(105,887; 93,193)	[0.017; 1.0]	871
	RD tree	(200,259; 175,037)	[0.012; 1.0]	1920
Hook	Octree	(28,426; 25,563)	[0.012; 1.0]	14
	RD tree	(34,709; 28,624)	[0.013; 1.0]	21
Hook2	Octree	(33,296; 30,212)	[0.011; 1.0]	17
	RD tree	(21,377; 17,616)	[0.013; 1.0]	35
Varco3	Octree	(53,646; 48,516)	[0.011; 1.0]	38
	RD tree	(49,901; 42,064)	[0.015; 1.0]	92

For CAD models, sharp features are preserved. Finally, mesh quality is improved. In the future we will test more models to make our code more robust and efficient.

Acknowledgements

This research was supported in part by Y. Zhang's ONR-YIP award N00014-10-1-0698 and an AFOSR grant FA9550-11-1-0346, which are gratefully acknowledged.

References

[1] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[2] M. S. Ebeida, A. Patney, J. D. Owens, and E. Mestreau. Isotropic conforming refinement of quadrilateral and hexahedral meshes using two-refinement templates. *Int. J. Numer. Meth. Engng*, 88(10):974–985, 2011.

[3] J. Edgel. An adaptive grid-based all hexahedral meshing algorithm based on 2-refinement. In *MS Thesis, Brigham Young University*, 2010.

[4] N. Folwell and S. Mitchell. Reliable whisker weaving via curve contraction. *Eng. Comput.*, 15(3):292–302, 1999.

[5] L. A. Freitag. On combining Laplacian and optimization-based mesh smoothing techniques. *Trends in Unstructured Mesh Generation, ASME*, 220:37–43, 1997.

[6] Y. Ito, A. Shih, and B. Soni. Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates. *Int. J. Numer. Methods Eng.*, 77(13):1809–1833, 2009.

[7] P. M. Knupp. Next-generation sweep tool: a method for generating all-hex meshes on two-and-one-half dimensional geometries. In *7th Int. Meshing Roundtable*, pages 505–513, 1998.

[8] X. Liang, M. Ebeida, and Y. Zhang. Guaranteed-quality all-quadrilateral mesh generation with feature preservation. *Comp. Meth. Appl. Mech. Engr.*, 199(29–32):2072–2083, 2010.

[9] X. Liang and Y. Zhang. Hexagon-based all-quadrilateral mesh generation with guaranteed angle bounds. *Comp. Meth. Appl. Mech. Engr.*, accepted, 2011.

[10] J. Qian and Y. Zhang. Automatic unstructured all-hexahedral mesh generation from B-Reps for non-manifold CAD assemblies. *Engineering with Computers*, DOI: 10.1007/s00366-011-0232-z, 2012.

[11] R. Schneiders. Refining quadrilateral and hexahedral element Meshes. In *5th Int. Meshing Roundtable*, pages 383–398, 1996.

[12] R. Schneiders, R. Schindler, and F. Weiler. Octree-based generation of hexahedral element meshes. In *5th Int. Meshing Roundtable*, pages 205–216, 1996.

[13] M. L. Staten, R. A. Kerr, S. J. Owen, and T. D. Blacker. Unconstrained paving and plastering: progress update. In *15th Int. Meshing Roundtable*, pages 469–486, 2006.

[14] Y. Zhang and C. Bajaj. Adaptive and quality quadrilateral/hexahedral meshing from volumetric Data. *Comput. Meth. Appl. Mech. Eng.*, 195(9–12):942–960, 2006.

[15] Y. Zhang, C. Bajaj, and B.-S. Sohn. 3D finite element meshing from imaging data. *Comput. Meth. Appl. Mech. Eng.*, 194(48–49):5083–5106, 2005.

[16] Y. Zhang, C. Bajaj, and G. Xu. Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. *Commun. Numer. Meth. Eng.*, 25(1):1–18, 2009.

[17] Y. Zhang, G. Xu, and C. Bajaj. Quality meshing of implicit solvation models of biomolecular structures. *Computer Aided Geometric Design*, 23(6):510–530, 2006.